

General Disclaimer

One or more of the Following Statements may affect this Document

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.
- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.
- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.
- This document is paginated as submitted by the original source.
- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

Genetic Algorithm for Optimization: Preprocessing with n Dimensional Bisection and Error Estimation

S.K. Sen

Department of Mathematical Sciences
Florida Institute of Technology
150 West University Boulevard
Melbourne, Florida 32901-6975, U.S.A.
sksen@fit.edu

and

Gholam Ali Shaykhian

Application, Simulation and Support Software Branch
Shuttle Processing Directorate
National Aeronautics and Space Administration
Kennedy Space Center, Florida 32899, U.S.A.
Ali.Shaykhian@nasa.gov

Abstract A knowledge of the appropriate values of the parameters of a genetic algorithm (GA) such as the population size, the shrunk search space containing the solution, crossover and mutation probabilities is not available a priori for a general optimization problem. Recommended here is a polynomial-time preprocessing scheme that includes an n -dimensional bisection and that determines the foregoing parameters before deciding upon an appropriate GA for all problems of similar nature and type. Such a preprocessing is not only fast but also enables us to get the global optimal solution and its reasonably narrow error bounds with a high degree of confidence.

1. Introduction

What is GA and how is it related to GP and EP A genetic algorithm (GA) inspired by Darwin's theory of evolution and employed to solve optimization problems — unconstrained or constrained — uses an evolutionary process. It is a search algorithm based on mechanics of natural selection and natural genetics. A GA is always randomized/probabilistic and polynomial time while the concerned search problem could be polynomial-time or NP-complete. It is a machine learning model which metaphorically derives its behavior from the processes of natural evolution. The GA for a general search problem starts with creation of a population of individuals, represented by chromosomes, which is a set of bit/character (non-bit) strings analogous to what we see in our own DNA. The individuals in the population then go through a process of evolution. Those individuals that are fitter/stronger, while competing for resources in the environment, are more likely to survive and propagate their genetic material (genome). Encoding of genome admits *asexual reproduction* (budding) resulting in genetically *identical* (to the parent) offspring while *sexual reproduction* allows genetically *radically different* offspring retaining, however, the general characteristics (species). At the molecular (building block) level, a pair of chromosomes bump into one another, exchange chunks of genome, which GA refers to as crossover operation. The crossover operation

depends on how good the individual (chromosome) is at competing in its environment. There are several nonconventional crossover operators such as the order-based crossover (OBC), the modified order crossover [31]. Among conventional crossovers, partially mapped (PMX), order (OX), and cycle (CX) crossovers [36] are a few. Each one has its own advantages. Some GAs use a simple fitness function model to select stochastically¹ individuals to undergo genetic operations such as crossover (genetically different offspring) or asexual reproduction (genetically unaltered offspring). This selection is called *fitness-proportionate* selection. Other GAs use a model in which certain randomly selected individuals in a subgroup compete and the fittest is selected. This selection is called *tournament selection*. A GA usually employs the process of fitness proportionate reproduction, which is near optimal in some statistical sense. Besides the reproduction and crossover operators, the third operator often used in a GA is *mutation* (sometimes called *background* operator) which changes a bit/character in a chromosome in a random way and is used usually sparingly although sometimes may play a dominant role. As a simulation of a genetic process, a GA is a motivated (fitness-based) random (stochastic) search rather than simply a random search and thus performs significantly better for optimization and monotonically improves the result in each successive generation. This character of a GA is unlike any algorithm based only on nonmotivated random search and is thus distinctly nonrandom in approaching towards the fittest chromosome/individual.

Simple number (bit or non-bit) string (representing a chromosome) manipulation operations permit implementation of (i) fitness proportionate reproduction, (ii) crossover, (iii) mutation, and (iv) other operations which may be designed depending on the requirement of the problem. While most of the work with GAs is based on fixed-length number strings, there are GAs that use variable-length number strings to represent chromosomes (individuals).

GP: how is it related to GA The fixed-lengthness and encoding the representation of the required solution as a character string is specific to a GA while a genetic programming (GP) does not have a fixed-length representation nor does it have encoding of the problem as strings of numbers. Genetic Programming (GP), functionally a branch of GA, differs from GA in the representation of the solution. GP creates computer programs (CPs) in computer languages as the solution while GA creates a string of numbers as the solution. Similar to a GA, GP uses three steps to solve problems: (i) Generate an initial population of random compositions of the *functions and terminals* of the problem (CPs as parse trees rather than lines of code). (ii) Execute each CP in the population and assign it a fitness value depending on how well it solves the problem. (iii) Generate a new population of CPs by (a) copying the best existing CPs, (b) creating new CPs by mutation, and (c) producing new CPs by crossover. The output of a GP is the best CP that appeared in any generation (the best-so-far solution) [1]. Since GP has much to do with theoretical computer science, we will here refrain from any further discussion on GP.

EP: how is it related to GA The evolutionary programming (EP), similar to evolutionary strategies (ES), differs from GA in two ways.

¹ The stochastic process in nature is strictly not random since it does not violate any law of nature or of natural optimization and its outcome is completely determined by natural process and is thus deterministic. It may be at best called pseudo-random (pseudo meaning false). Artificial (implemented on a computer or implemented through a mechanical process such as the tossing a coin, throwing a die and rotating a roulette wheel) stochastic process is strictly also not random since the output/numbers generated in any of these ways must have followed a procedure and thus are related to the inputs through the procedure.

- (i) A GA encodes/represents 'the problem solutions as a set of strings of numbers/characters while an EP has no such constraint. In an EP, the representation of problem solutions varies from one problem to another.
- (ii) In GA, two solutions are mated to form new solutions (offsprings) through crossover/mutation, while in an EP, each member of the population generates an offspring by *mutation operators* only. Typically EP does not use crossover. Take, for example, the following two mutation operations.

Mutation operation 1

Parent solution:	1, 3, 7, 5, 6, 2, 8, 4, 9
Selected Positions (random):	
Off-spring (new solution):	1, 3, 8, 5, 6, 2, 7, 4, 9
(selected positions interchanged)	

Mutation operation 2

Parent solution:	1, 3, 7, 5, 6, 2, 8, 4, 9
Cut points (random)	
Off-spring (new solution)	1, 3, 8, 2, 6, 5, 7, 4, 9
(cut positions reversed)	

There is a new mutation operator that performs random jumps within the neighborhood of a local optimum along with a new replacement technique that tries to avoid the emergence of highly similar or identical members of the population [41]. One may thus define many mutation operations and experiment in many ways for a specified type of problems. Any one or more (maybe 2, 3, . . .) mutation operations may be performed one or more times in a single algorithm.

EP does not emulate like GA specific genetic operations (viz., reproduction, crossover, mutation) as observed in nature (biology). It is a stochastic optimization strategy similar to GA. Like both ES and GA, EP is a useful optimization method when other methods such as the gradient ascent/descent techniques (calculus-based and computed analytically/numerically) and enumerative methods are not feasible/tractable (e.g., violently fluctuating one or more variable functions). For further information on EP, refer [2]).

Why preprocessing We are accustomed to run a program and get the required result in the very first run. Is it absolutely necessary to get the most desired result in the very first run? What happens if we do not get the most desired result or if we do not know if the result is truly the most desired result? Now that computing resources are available aplenty most of which are unutilized in terms of the PC (computer) system remaining idle most of the time, it is most often insignificant to talk about computing cost for most real world problems. In this context, preprocessing in terms of multiple runs of the program with appropriate change in parameters in successive runs to get the best result is desired. This is particularly so for randomized algorithms such as the genetic algorithms. In real time programs/algorithms, however, such a preprocessing is undesirable when particularly the time constraint is severe. A GA is (locally non-deterministic) randomized. Unlike any deterministic algorithm, the output result will go on changing for the same fixed input parameters since the required random number generation (RNG) will practically always be different. We generally will never force RNG from the same seed(s) every time we run the same program. For instance, in Matlab, one may use the

command “rand(‘state’, 0)” to get back the most recently generated random numbers. Unless one likes to see/get back the previous run result, one may not use this command. However, to obtain quality solution with a measure of how good the solution is in a comparable polynomial-time computational complexity, preprocessing before deciding on the several vital parameters used to solve a given optimization problem is stressed upon here. The problem may be NP-complete such as the traveling salesman problem (TSP) and general integer programming problems (IPs). It may also be polynomial-time such as the unconstrained optimization of a violently fluctuating function having too many local minima/maxima. The deterministic algorithm for the TSP for n cities will need $(n - 1)!$ path searches and is combinatorial (exponential) where $f_n = (n - 1)! \approx \sqrt{(2\pi/n)} (n/e)^n$ by Sterling formula and thus intractable for even $n = 100$. When $n = 100$ and the computer executes 10^{10} , i.e., 10 billion floating point operations (flops) per second then to deterministically solve the problem, we need to execute 9.3248×10^{155} flops. Consequently, we would require 2.9564×10^{135} centuries compared to which, even the estimated age of the universe is a numerical zero [5]. The ill-posed (violently fluctuating) multivariable function optimization, though theoretically tractable through a deterministic polynomial algorithm such as the steepest descent/ascent method (possibly in an extremely small subspace) is often infeasible computationally. In both these cases, GA, which is a motivated random search algorithm, is very useful and produces the solution in a polynomial-time.

We certainly like to have as good a global optimal solution (GOS) as possible and as quickly as possible. We would also certainly like to have some satisfying measure/estimate of how good the GOS is so that we can sufficiently confidently say about the relative quality. Keeping in mind these two requirements, we advocate a systematic study of the genetic parameters such as population size, length of each member (chromosome) of the population, types of reproduction, crossover, and mutation operations, interval of each dimension of an n -dimensional space (domain of the function), number of subintervals for each of these intervals and number of runs. Such a study usually does not change a polynomial time algorithm to an exponential-time one nor is it expected to produce an increase of unacceptable time of computation. On the other hand, this study will provide not only a deeper insight of the given problem but also fairly good (not necessarily optimal) values of the foregoing parameters. It is not necessary to be too strict to obtain best values of the concerned parameters, which is not only unacceptably time consuming but also not required so far as the solution is concerned. Preprocessing involves this study and provides us a much better feel about the problem. *Trying to solve a problem just by running a given computer program without having practically any knowledge of the character/nature of the problem is not desirable.* This is more so if the problem is truly ill-conditioned with respect to the solution procedure.

We are not in general concerned with well-posed problems. We are concerned with ill-posed both large and small problems for which we propose pre-processing for the purpose of deciding on the parameter values discussed in the foregoing section. If we do not do so, we might miss the GOS without even knowing whether our solution is reasonably acceptable. Premature convergence due to reduction of the diversity and evolutionary stagnation is also a possible problem. Our preprocessing includes, among other features, idea similar to multi-step GA suggested for such a problem [25].

The knowledge that we acquire through preprocessing and that tells us the behavior of the function and quality of the solution will be absent if we do not give due importance to preprocessing. The need for preprocessing is more pronounced for randomized algorithms such

as the GAs than for deterministic ones. This is because, unlike the deterministic algorithms/programs, the GAs will produce different results in different runs of the same program. Our goal is to reduce/minimize the difference/distance between the best GOS and the worst GOS in a run not only to improve our confidence on the quality of the result but also to eliminate the possibility of missing the GOS. The references [1-45] cited here are a rich source of information in the genetic/evolutionary approach and the related areas and are of use to those who desire to do research in these areas.

In section 2, we discuss the preprocessing involving the various genetic parameters, operators, their perturbations, and their scope before finalizing the GA for specific type of multi-dimensional optimization problems both constrained and unconstrained. Section 3 includes an example of maximization of two variable rapidly fluctuating functions with numerical error estimation and the time complexity. Section 4 comprises conclusions.

2. Preprocessing with *n-D Bisection*: Parameters, Operators, and Perturbations

We first identify all the parameters and the genetic operators. We then decide on a plan how to perturb the parameters so that the distance between two solutions, viz., ($\| \text{best GOS} - \text{worst GOS} \|$ in each subspace) comes down as much as possible. These perturbations will give us deeper insight into the character/nature and the possible location of the GOS. The parameters are (i) the n dimensional search space, (ii) the size of a member (chromosome) of the population, i.e., size of a solution vector having n elements, and (iii) the size of the population, i.e., the number of candidate solution vectors. The genetic operators that we advocate to perturb here are (i) crossover and (ii) mutation. There are many possible ways of implementing/perturbing these operators and deciding their probabilities based on the outcome/result. The sole motive of any perturbation in probability of crossover, probability of mutation, or population size is to increase the fitness value, i.e., to proceed toward the actual GOS. In fact, the perturbations will also improve enormously our confidence in that the true GOS has not eluded and escaped from our search.

The parameter (ii), viz., the size of a member depends on the number of significant digits/decimal places accuracy for the solution and the given dimension n . Once we decide on the accuracy required/desired, we have no scope to perturb this parameter. For example, if the interval in one dimension/variable x is $[a, b]$ and if we need 7 decimal places accuracy then the domain of the variable x has length $L = (b - a)$ and the interval should be divided into at least $L \times 10^7$ equal size subintervals. If $b = 4$ and $a = -3$ then we need 27 bits for the binary vector (chromosome) since $67108864 = 2^{26} \leq 7 \times 10^7 \leq 2^{27} = 134217728$. The mapping from a binary string ($b_{26}b_{25} \dots b_1 b_0$) to a real (decimal) number x from the interval $[a, b]$ is $x = a + [x'(b - a)/(2^{27} - 1)]$, where $x' = (\sum_{i=0}^{26} b_i \times 2^i)_{10}$. Now if we have n variables then our binary vector (string) will be at least n times as large to represent a member (chromosome) of the population.

We are now left with two other parameters (i) and (iii), viz., the search space and the population size. Interestingly both these parameters are related. For the given search space, we can perturb the population size, possibly increasing step by step the size starting from an arbitrary small value to a large value so that the difference between the worst GOS and the best GOS tends to vanish. Or, we can keep the population size fixed (constant and not relatively large) throughout and divide the search/solution space into subspaces. We then apply the GA for each subspace and identify that subspace which would contain the required GOS assuming

the existence of only one GOS in the given original search space. The rest of the subspaces are removed. Thus our search space is significantly reduced. In case there are two or more doubtful subspaces where the GOS is likely, then we keep these subspaces while removing other subspaces. This later situation could arise if our population size is not sufficiently large. Although the genetic operators and their probabilities could contribute to some extent to our doubt, the main contributor is the population size. However, our main motive is to quickly and confidently reduce the search space and consequently we will keep population size unaltered throughout for any subspace and its further subdivision to track down the GOS. We will play, if necessary, with the genetic operators by varying their implementations and changing their probabilities to see if there is an improvement.

Preprocessing procedure for parameter values We first identify the parameters whose values we will determine by actually running the GA for the given problem a number of times. It is not out of place to mention that due to enormous processing power/speed² available to us, where most of this power is unutilized, focusing too much on relative time complexity (minor differences in time of computation) is wasteful. Executing a GA a number of times and using the results still keeps the complexity polynomial-time and is desirable.

Search space, population, and its member sizes Two parameters, viz., the population size and the size of the search space are vital. The size (number of bits) of a member (chromosome) of the population is predetermined based on the accuracy needed. However, if too much of accuracy is required, then this might either not be possible due to precision limitation of the computer or would increase the search density too much and consequently the time complexity. Further, real world implementation of these highly numerically accurate quantities may not be possible as any measuring device cannot, in general, give an accuracy more than .005% [5]. Hence, it is enough if our accuracy of a quantity to be used in the real world environment is just four significant digits. Thus we take the member size such that it gives at most four significant digit accuracy and at least that much accuracy (usually 2 to 4 significant digits) required in the given problem. Thus the member size is practically fixed unless the concerned quantity is not an intermediate one. For an intermediate quantity, however, one should have more than four significant digit accuracy so that the terminal quantity to be used in the physical world has four significant digit accuracy.

Three or less variable functions: Generating graphs to reduce given search space For a given multi dimensional constrained/unconstrained optimization problem involving real numbers, we assume that for the given problem, we are not having any knowledge about the character of the problem to start with. For one, two, and three dimensional problems (given in an analytical form, not in a tabular form) which may be considered small but nontrivial and may crop up in

² Every 18 months processor speed is doubling, every 12 months bandwidth is doubling while every 9 months hard disk space is doubling. Currently, in many commercially available computers, we may execute over a billion floating-point operations per second (flops). On the extreme as of now (2006) the processing speed has touched 36 billion flops. The storage (CPU registers, cache, main executable memory, hard disk) sizes and their retrieval speed also have proportionately increased and are increasing. Since for higher processing power, higher storage space is a must to avoid any bottle neck in overall processing speed., we do have terabyte auxiliary storage space currently. The next goal is to achieve peta flops (peta = 10^{15}) speed. Is there a limit beyond which the speed cannot be increased? Indeed there is a limit set by the speed of light barrier, thermal efficiency barrier, as well as quantum barrier do limit the computational power [5]. The number of protons in the universe is estimated to be around 10^{73} . If the whole universe is dedicated to information processing, then no more than 7.8996×10^{103} bits per year can be processed [5]. However, we are still too far away from these extreme speeds, which appear to be a universal maximum in silicon technology.

many practical applications, one may get the graph using the Matlab commands and get considerable information about the nature of the problem, e.g., about how frequently/closely the function is fluctuating. It may be noted that a graph is just a crude form/representation of the actual required numerical values. Thus the graph will at best tell us about search space and may help us reducing the search space for global optimization to an extent.

Four or more variable functions: bisecting space to reduce search space For four or more variable functions such a generation of graphs is impossible. We have two-dimensional paper that can be used to represent two-dimensional graphs. Since we, the common human beings, are capable of visualizing three-dimensional objects, a three dimensional graph drawn (using projective geometry) on a two-dimensional paper can be visualized. It may be interesting to note that we can imagine/visualize n -dimensional ($n \leq 3$) Cartesian coordinates having a point (in zero dimension), one straight line (in 1 dimension), two mutually perpendicular straight lines (in 2 dimensions), and 3 mutually perpendicular straight lines (in three dimensions). Can we imagine 4 (or more) mutually perpendicular straight lines (in 4 or more dimensions)? The answer is 'no'. However, mathematical models for most real world problems do involve n dimensions where n could be even over a million. In such an n -dimensional ($n \geq 4$) problem where graph representation is impossible, we divide the given search space into a number of subspaces and then use the GA for each subspace keeping the population size constant. That is, the original population size meant for the complete search space will also remain the same for each subspace. These searches will give us some kind of rough information about the position of the global optimal point. Consequently it will help us to reduce search space(s).

Estimation of the population size If we somehow know the number of fluctuations of a too frequently fluctuating function – single or multivariable – in the given space, then we may use the concept of the *sampling theorem* to decide an excellent population size. The theorem states that for a band-limited signal with maximum frequency f_{max} , called the Nyquist frequency, the equally spaced sampling frequency f_s must be greater than twice the maximum frequency f_{max} , i.e., $f_s > 2f_{max}$ so that the signal can be uniquely reconstructed without *aliasing*. If there are k fluctuations, then we may necessarily choose the population size $> 2k$. Unfortunately, this is often not possible/easy to be determined for most multidimensional functions. Further, assuming that the search space is large and the n -D function has just one or two fluctuations then this situation does not necessarily imply that we may take a population size 4 or just a few more and get good result. We take a reasonable population size, say, 128 for the problem and keep this fixed throughout irrespective of any knowledge of the number of fluctuations.

n -D successive bisection of the search space Let the problem be to find out the GOS for the function $f(x_1, x_2 \dots x_n)$. Let the n -D search space be $x_i \in [\alpha_i, \beta_i]$, $i = 1(1)n$.

Step 1 Choose the size of each member (chromosome) of the population as $n \times 27$ (if $\alpha_i = -3$, $\beta_i = 4$ for all i and 7 decimal places accuracy is desired).

Step 2 Choose the population size as 128 (arbitrarily).

Step 3 Bisect each $[\alpha_i, \beta_i]$, $i = 1(1)n$. This will produce 2^n subspaces. Run, for each subspace, a GA with the fixed population size 128 and with the number of generations 64 (arbitrarily chosen).

Step 4 Record, for each subspace, (i) the best GOS, (ii) the worst GOS and (iii) their distance/difference $\| \text{best GOS} - \text{worst GOS} \|$. If the distance D for the subspace containing the most desired best GOS is sufficiently small then the required GOS is reached, the relative error estimate is $E_r = D / \| \text{most desired best GOS} \|$, terminate.

Step 5 Retain only that subspace(s) including the doubtful ones which has/have most desired best GOS. Replace the search space by this subspace and repeat Steps 3 to 4.

Before starting the successive bisection, we may run the GA for the given original search space and determine the distance. The distance is not expected to be small. However, even if this distance is small, we proceed to the bisection stage to be sure/confident that we have not missed the required GOS. While we have so far focused on the n -D unconstrained function optimization, the foregoing procedure can be adapted to constrained nonlinear optimization.

The foregoing bisection-based preprocessing outputs sufficiently narrow subspace, which can be explored further by perturbing the genetic operators, viz., crossover and mutation and their probabilities. This will enable us to obtain the reduced search space for problems of the same type as well as appropriate genetic operators with appropriate probabilities for these problems.

The probability of overlooking the actual GOS is practically eliminated in this preprocessing since we can perturb (increase or decrease) the population size and proceed with the same n -D bisection keeping all the other operators/parameters unchanged and obtain the same result. If we do not get the same result, which is unlikely, it will immediately prompt us to increase the population size. Such a prompting may not at all happen. *The most important gain of preprocessing is our extremely high level of confidence in the quality of the computed GOS.*

Different implementations of GA operators and their probabilities The classical GA operators are crossover and mutation. Each of these two operators may be implemented in very many ways, where each way produces different outputs. The probability of each of these two operators can be chosen and this will also result in different outputs. The information that these outputs provide us is of immense use to us in terms of deciding the implementation and the probabilities. Further the scope of the true GOS escaping from our search and a wrong GOS getting into our net tends to vanish.

3. Example

To preprocess for the global maximum of the two variable function

$$f(x_1, x_2) = x_1 \sin(4\pi x_1) + x_2 \sin(20\pi x_2) + 17.3 \text{ given the search space as}$$

$\{(x_1, x_2) | \alpha_1 \leq x_1 \leq \beta_1, \alpha_2 \leq x_2 \leq \beta_2\}$, where $\alpha_1 = -3, \beta_1 = 15, \alpha_2 = 3, \beta_2 = 6$, we bisect the search space into $2^2 = 4$ subspaces, viz., S1, S2, S3, and S4, where

$$\begin{aligned} \text{S1: } \{(x_1, x_2) | -3 \leq x_1 \leq 6, 3 \leq x_2 \leq 4.5\}, \text{ S2: } \{(x_1, x_2) | -3 \leq x_1 \leq 6, 4.5 \leq x_2 \leq 6\}, \\ \text{S3: } \{(x_1, x_2) | 6 \leq x_1 \leq 15, 3 \leq x_2 \leq 4.5\}, \text{ S4: } \{(x_1, x_2) | 6 \leq x_1 \leq 15, 4.5 \leq x_2 \leq 6\}. \end{aligned}$$

For each subspace we run the GA 5 times. Tableau³ 1 provides us, for each subspace, the best GOS (here global maximal solution) out of 5 runs, the worst GOS out of 5 runs, and their Euclidean distance $\|\text{best GOS} - \text{worst GOS}\|$. Suppose that we represent x_1 and x_2 up to 7

³ Instead of the term "Table" we write "Tableau" as we do for solving a linear program using the simplex method. While Table has entries static, Tableau has entries dynamic (i.e., entries go on changing).

decimal digits. Since $-3 \leq x_1 \leq 6$, x_1 will be represented by a string of 27 bits. Similarly, since $3 \leq x_2 \leq 6$, x_2 will be represented by a string of 25 bits.

Depending on the values of the best GOS and the worst GOS for each subspace, we select that subspace(s) for which the best GOS value(s) are distinctly considerably higher than the best GOS values of other subspaces. Replace the search space by the selected subspace(s), execute the GA with the new reduced search space and obtain/recompute the entries of Tableau 1 to produce Tableau 2. From Tableau 1, we find here that Subspace 4 is the best subspace in which true GOS is expected to exist. Hence we will replace the search space by Subspace 4 and generate Tableau 2 exactly in the same way as we did for Tableau 1.

Tableau 1 Number of runs is 5. Best GOS out of 5 runs, worst GOS out of 5 runs, and the distance between them for each subspace (Population size is 128, each member of the population is $27 + 25$, i.e., 52 bits long, and number of generations in each run is 60.)

	Best GOS out of 5 runs and $f(x_1, x_2)$	Worst GOS out of 5 runs and $f(x_1, x_2)$	Distance Best GOS–Worst GOS
Subspace 1 ($-3 \leq x_1 \leq 6$, $3 \leq x_2 \leq 4.5$)	$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5.6230939 \\ 4.4252964 \end{bmatrix}$ $f(x_1, x_2) = 27.3460101$	$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5.6572269 \\ 4.2250205 \end{bmatrix}$ $f(x_1, x_2) = 26.7246432$	0.2031637
Subspace 2 ($-3 \leq x_1 \leq 6$, $4.5 \leq x_2 \leq 6$)	$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5.6265164 \\ 5.9262224 \end{bmatrix}$ $f(x_1, x_2) = 28.8342451$	$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5.1268724 \\ 5.8228386 \end{bmatrix}$ $f(x_1, x_2) = 28.1946774$	0.5102277
Subspace 3 ($6 \leq x_1 \leq 15$, $3 \leq x_2 \leq 4.5$)	$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 14.6247697 \\ 4.4258769 \end{bmatrix}$ $f(x_1, x_2) = 36.3438695$	$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 13.6306948 \\ 4.4244118 \end{bmatrix}$ $f(x_1, x_2) = 35.3171967$	0.9940760
Subspace 4 ($6 \leq x_1 \leq 15$, $4.5 \leq x_2 \leq 6$)	$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 14.6322016 \\ 5.9248899 \end{bmatrix}$ $f(x_1, x_2) = 37.7970724$	$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 14.1320801 \\ 5.7250442 \end{bmatrix}$ $f(x_1, x_2) = 37.1012047$	0.5385720

Continuing we obtain at the fifth successive bisection, the GOSs as

$$\text{Best GOS} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 14.6254170 \\ 5.9250368 \end{bmatrix}, \text{ worst GOS} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 14.6269800 \\ 5.9250421 \end{bmatrix},$$

and the corresponding function values are $f(x_1, x_2) = 37.8502372$, $f(x_1, x_2) = 37.8474740$, respectively. The distance $d = ||\text{Best GOS} - \text{Worst GOS}|| = 5.6302495e-004$. The relative error is $d/||\text{Best GOS}|| = 5.6302495e-004/15.7800153 = 3.5679620e-005$. The result is correct at least up to 4 significant digits.

However, to compare the time complexity we take two different number of generations, viz., 60 and 30 and keep number of successive 2-D bisections = 3 for both these numbers. For the number 60, the time complexity is 202.641 seconds while for 30, it is 92.793 seconds (less

than half the time for half the number of generations) implying that the computational complexity will increase with the increase of the number of generations.

This successive 2-D bisection will result in the distance between the best GOS and the worst GOS tending to vanish. Based on the accuracy desired, we terminate after noting down the best GOS of the best subspace (i.e., the subspace having the desired best GOS).

Tableau 2 Number of runs is 5. Best GOS out of 5 runs, worst GOS out of 5 runs, and the distance between them for each subspace (Population size is 128, each member of the population is $27 + 25$, i.e., 52 bits long, and number of generations in each run is 60.)

	Best GOS out of 5 runs and $f(x_1, x_2)$	Worst GOS out of 5 runs and $f(x_1, x_2)$	Distance $ \text{Best GOS} - \text{Worst GOS} $
Subspace 1 ($6 \leq x_1 \leq 10.5$, $4.5 \leq x_2 \leq 5.25$)	$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 10.1209674 \\ 5.2250521 \end{bmatrix}$ $f(x_1, x_2) = 32.6329990$	$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 9.6232693 \\ 5.2280276 \end{bmatrix}$ $f(x_1, x_2) = 32.0547111$	0.4977070
Subspace 2 ($6 \leq x_1 \leq 10.5$, $5.25 \leq x_2 \leq 6$)	$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 10.1256415 \\ 5.9250454 \end{bmatrix}$ $f(x_1, x_2) = 33.3503338$	$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 10.1256875 \\ 5.7260751 \end{bmatrix}$ $f(x_1, x_2) = 33.1383259$	0.1989703
Subspace 3 ($10.5 \leq x_1 \leq 15$, $4.5 \leq x_2 \leq 5.25$)	$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 14.6253576 \\ 5.2250476 \end{bmatrix}$ $f(x_1, x_2) = 37.1502342$	$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 14.1145018 \\ 5.2265958 \end{bmatrix}$ $f(x_1, x_2) = 36.4921991$	0.5108581
Subspace 4 ($10.5 \leq x_1 \leq 15$, $5.25 \leq x_2 \leq 6$)	$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 14.6253645 \\ 5.9246063 \end{bmatrix}$ $f(x_1, x_2) = 37.8480049$	$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 14.1301976 \\ 5.9250424 \end{bmatrix}$ $f(x_1, x_2) = 37.3250894$	0.4951671

Although we have not included the perturbation of other parameters as a part of preprocessing, which is not required for this particular problem, we will certainly do so for much more ill-conditioned problems and track the true GOS with sufficiently high confidence. We have not included more involved violently fluctuating large dimensional problems to conserve space although our preprocessing for such test problems has been excellently successful. The concerned Matlab program is omitted also to conserve space.

4. Conclusions

Preprocessing is our recommendation Our focus is on preprocessing and not so much individually on the n-D bisection, and genetic parameters. We explore the ins and outs of the optimization problem changing population size, number of generations, number of runs as well as implementing the genetic operators in different ways including varying their probabilities. We recommend preprocessing and this will enormously increase our confidence on the quality of the result and this will enable us to compute a convincing and reasonable error-bound. This bound is the scientific means of relying on the result and accepting it. Further the complexity

due to preprocessing never becomes exponential. It remains polynomial-time just like any GA without preprocessing. In view of enormous personal computing available to us and most of which remains unutilized and hence a waste, preprocessing becomes attractive and a necessity in most ill-posed problems.

Constrained optimization We have discussed only unconstrained multi-variable function optimization here. The preprocessing concept discussed here can be straightway implemented for constrained optimization problems too. We will discuss this in a future paper.

Escape of true GOS from our preprocessing search tends to vanish It is not uncommon that a wrong GOS gets caught in our search net if the recommended preprocessing is not used. Preprocessing provides us a wealth of valuable information into the very nature of the function. Without this, we will have very few information to fall back on to believe that the caught GOS is the right GOS. The very nature of random number generation, both quasi- and pseudo-, by various methods [21, 37] also is helpful in our perturbation, which is central to our preprocessing.

References

1. Koza, J.R. , Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, Massachusetts, 1992.
2. Fogel, D.B., "Evolutionary Computation: Toward a New Philosophy of Machine Intelligence," IEEE Press, Piscataway, NJ. 1995.
3. Goldberg, D.E., Genetic Algorithms: In Search, Optimization & Machine Learning, Addison-Wesley, Boston, 1989.
4. Zbigniew, M., Genetic Algorithms + Data Structures = Evolution Programs, Springer-Artificial Intelligence, 1998.
5. Lakshmikantham, V.; Sen, S.K. Computational Error and Complexity in Science and Engineering, Elsevier, Amsterdam, 2005.
6. Krishnamurthy, E.V. and Sen, S.K., Introductory Theory of Computer Science, Affiliated East-West Press, New Delhi, 2004.
7. Tsai, H; Yang, J.;Tsai, Y.;Kao, C., An evolutionary algorithm for large traveling sales man problems, IEEE Trans. (SMC), Part B: Cybernetics, 34, 4, August 2004, 1718-1729.
8. Chakraborty, M.; Chakraborty, U.K., Applying genetic algorithm and simulated annealing to a combinatorial optimization problem, International Conference on Information, Communications and Signal Processing (ICICS'97), Singapore, 9-12 Sep 1997, 929-933.
9. Adler, D., Genetic algorithm and simulated annealing: A marriage proposal, 0-7803-0999-5/93/\$03.00C1993 IEEE, 1104-1109.
10. Pasiadis, V.;Karras, A.D.; Papademetriou, R.C.,Traffic engineering in multi-service networks comparing genetic simulated annealing optimization techniques, 0-7803-8359-1/04/\$20.00C2004 IEEE, 2325-2330.

11. Krishnakumar, K.; Swaminatham, R.; Garg, S.; Narayanaswamy, S., Solving large parameter optimization problems using genetic algorithms, AIAA-95-3223-CP, 1995, 449-460.
12. Ghoshray, S.; Yen, K.K., More efficient genetic algorithm for solving optimization problems, 0-7803-2559-1/95 \$4.00 C 1995 IEEE, 4515-4520.
13. Lee, K.Y.; Yang, F.F., Optimal reactive power planning using evolutionary algorithms: A comparative study for evolutionary programming, evolutionary strategy, genetic algorithm, and linear programming, IEEE Trans. (Power Systems), 13, 1, Feb 1998, 101-108.
14. Nguyen, H.D.; Yoshihara, I.; Yasunaga, M., Modified edge recombination operators of genetic algorithms for the traveling salesman problem, 0-7803-6456-2/00/\$10.00 C 2000 IEEE, 2815-2820.
15. Takenaka, Y.; Funabiki, N., An improved genetic algorithm using the convex hull for traveling salesman problem, 0-7803-4778-1/98 \$10.00 C 1998 IEEE, 2279-2284.
16. Horng, J.; Kao, C.; Chen, G., An error-tolerance genetic algorithm for traveling salesman problems, 0-7803-1-95 @4.00 C 1995 IEEE, 795-799.
17. Nagata, Y.; Kobayashi, S., An analysis of edge assembly crossover for the traveling salesman problem, 0-7803-5731-0/99 \$10.00 C 1999 IEEE, III-628-III-633.
18. Gunnels, J.; Cull, P.; Holloway, J.L., Genetic algorithms and simulated annealing for gene mapping, 0-7803-1899-4/94 @4.00 C 1994 IEEE, 385-390.
19. Maekawa, K.; Mori, N.; Tamaki, H., A genetic solution for the traveling salesman problem by means of athermodynamical selection rule, 0-7803-2902-3/96/ \$4.00 C 1996 IEEE, 529-534.
20. Jellouli, O., Intelligent dynamic programming for the generalized traveling salesman problem, 0-7803-7087-2/01/\$10.00 C 2001 IEEE, 2765-2768.
12. Thonemann, U.W., Finding improved simulated annealing schedules with genetic programming, 0-7803-1899-4/94 \$4.00 C 1994 IEEE, 391-395.
21. Sen, S.K.; Samanta, T.; Reese, A., Quasi- versus pseudo-random generators: Discrepancy, complexity and integration-error based comparison, International Journal of Innovative Computing, Information and Control, 2, 3, 2006, 1-31.
22. Schmitendorf, W.E.; Shaw, O.; Benson, R., Using genetic algorithms for controller design: Simultaneous stabilization and eigenvalue placement in a region, AIAA-92-4465-CP, 1992, 757-761.
23. White, C. M.; Yen, G.G., A hybrid evolutionary algorithm for traveling salesman problem, 0-7803-8515-2/04/\$20.00 C IEEE, 1773-1478.
24. Boomsma, W., Using adaptive operator scheduling on problem domains with an operator manifold: Applications to the traveling salesman problem, 0-7803-7804-0/03/\$17.00 C 2003 IEEE, 1274-1279.
25. Watabe, H. and Kawaoka, T., Application of multi-step GA to the traveling salesman problem, 0-7803-6400-7/00/\$10.00 C 2000 IEEE, 510-513.
26. Ho, S.; Chen, J., A GA-based systematic approach for solving traveling salesman problem using an orthogonal array crossover, 0-7695-0589-2/00 \$10.00 C 2000 IEEE, 659-663.
27. Merz, P.; Freisleben, B., Genetic local search for the TSP: New results, 0-7803-3949-5/97/\$10.00 C 1997 IEEE, 159-164.

28. Talbi, H.; Draa, A.; Batouche, M., A new quantum-inspired genetic algorithm for solving the traveling salesman problem, 0-7803-8662-0/04/\$20.00 C 2004 IEEE, 1192-1197.
29. Wei, J.; Lee, D.T., A new approach to the traveling salesman problem using genetic algorithms with priority encoding, 0-7803-8515-2/04/\$20.00 C 2004 IEEE, 1457-1464.
30. Zhang, L.; Wang, L., Genetic ordinal optimization for stochastic traveling salesman problem, 0-7803-8273-0/04/\$20.00 C 2004 IEEE, 2086-2090.
31. Ray, S.S.; Bandyopadhyay, S.; Pal, S.K., New operators of genetic algorithms for traveling salesman problem, Proc. 17th International Conference on Pattern Recognition (ICPR'04) 1051-4651/04 \$20.00 IEEE, IEEE Computer Society.
32. Tsai, C.; Tsai, C., A new approach for solving large traveling salesman problem using evolutionary ant rules, 0-7803-7278-6/02/\$10.00 C 2002 IEEE, 1540-1545.
33. Baraglia, R.; Hidalgo, J.I.; Perego, R., A hybrid heuristic for the traveling salesman problem, IEEE Trans. (Evolutionary Computation), 5, 6, Dec 2001, 613-622.
34. Lee, I.; Sikora, R.; Shaw, M.J., A genetic algorithm based approach to flexible flow-line scheduling with variable lot sizes, IEEE Trans. (SMC), Part B: Cybernetics, 27, 1, Feb 1997, 36-54.
35. Xuan, W.; Li, Y., Solving traveling salesman problem by using a local evolutionary algorithm, 0-7803-9017-2/05/\$20.00 C IEEE, 318-321.
36. Tagawa, K.; Kanzaki, Y.; Okada, D., A new metric function and harmonic crossover for symmetric and asymmetric traveling salesman problems, 0-7803-4869-9/98 \$10.00 C 1998 IEEE, 822-827.
37. Lakshmikantham, V.; Sen, S.K.; Samanta, T., Comparing random number generators using Monte Carlo integration, Internl. J. Innovative Computing, Information and Control, 1, 2, June 2005, 143-165.
38. Pullan, W., Adapting the genetic algorithm to the traveling salesman problem, 0-7803-7804-0/03/\$17.00 C 2003 IEEE, 1029-1035.
39. Wang, L.; Maciejewski, A.A.; Siegel, H.J.; Roychowdhury, V.P., A comparative study of five parallel genetic algorithms using the traveling salesman problem, 1063-7133/98 \$10.00 C 1998 IEEE, 345-34.
40. Jin, H.; Leung, K.; Wong, M.; Xu, Z., An efficient self-organizing map designed by genetic algorithms for the traveling salesman problem, IEEE Trans. (SMC), Part B: Cybernetics, 33, 6, Dec 2003, 877-888.
41. Freisleben, B.; Merz, P., A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems, 0-7803-2902-3/96/ @4.00 C 1996 IEEE, 616-621.
42. Hajela, P.; Lin, C., Genetic search strategies in multicriterion optimal design, AIAA-91-1040-CP, 1991, 354-363.
43. Hassan, R.; Crossley, W., Variable population-based sampling for probabilistic design optimization with genetic algorithm, AIAA 2004-0452, 2004, 1-22.
44. Hassan, R.; Crossley, W., Discrete design optimization under uncertainty: A generalized population-based sampling genetic algorithm, AIAA 2004-1586, 2004, 1-18.
45. KrishnaKumar, K., Genetic algorithms: An introduction and an overview of their capabilities, AIAA-92-4462-CP, 1992, 728-738.